A Study of TCP Dynamics over HFC Networks

Omar Elloumi *
Omar.Elloumi@enst-bretagne.fr

Nada Golmie **
golmie@isdn.ncsl.nist.gov

Hossam Afifi *
Hossam.Afifi@enst-bretagne.fr

David Su **
dsu@isdn.ncsl.nist.gov

* Ecole Nationale Supérieure des Télécommunications de Bretagne, Networks and Multimedia Department
2 rue de la Chataigneraie, BP 78, 35512 Cesson Sévigné, France

** National Institute of Standards and Technology
Gaithersburg, Maryland 20899, USA

Abstract

New broadband access technologies such as Hybrid Fiber Coaxial (HFC) are likely to provide fast and cost effective support to a variety of applications including Video on demand (VoD), interactive computer games, and internet-type applications such as Web browsing, ftp, email, and telephony. Since most of these applications, use TCP as the transport layer protocol, the key to their success largely depends on the effectiveness of the TCP protocol. We investigate the performance of TCP in terms of effective throughput in an HFC network environment using different load conditions and network buffer sizes. We find that TCP experiences low throughput as a result of the well known problem of ACK compression. An algorithm that controls ACK spacing is introduced to improve TCP performance.

Keywords*: TCP, HFC, ATM, Congestion avoidance, ACK compression.*

1 Introduction

The emergence of the HFC technology has a significant impact on already deployed Cable TV networks. As a return path from the stations to the headend becomes available, Cable network operators are able to add more services to television broadcast. A Medium Access Control (MAC) layer protocol is implemented at the root (or headend) and at each of the cable network nodes (or stations) to allow various nodes to share resources in a multiaccess environment. It also controls the upstream (from the stations to the headend) and the downstream (from the headend to the stations) link transmissions. MAC protocol specifications are being drafted by the IEEE 802.14 working group to accomodate the needs of current and future network applications. The IEEE 802.14 Draft document [5] contains various MAC defining characteristics such as: frame format, station addressing, timing and synchronization procedures, and the ternary-tree mechanism to resolve collisions resulting from two or more stations transmitting at the same time. The MAC draft also provides the necessary "hooks" to support higher layer services such as CBR, VBR

Omar Elloumi will serve as the corresponding author, Phone: (33) 2 99 12 70 42, Fax: (33) 2 99 12 70 30
This work was done while Omar Elloumi was a guest researcher at NIST in Spring 1997.
A complete version of this paper is available via anonymous ftp from ftp.rennes.enst-bretagne.fr pub/reseau/elloumi/tcp_hfc.ps

and ABR services for ATM. Numerous performance evaluation studies have been conducted on MAC protocol elements such as contention resolution and bandwidth allocation [8]. Also, some preliminary work has been presented on improving the ABR service over HFC in [9]. But so far, little work has been done in studying the details and evaluating the performance of the TCP protocol in an HFC network environment.

The performance of TCP in networks with slow ACKs channel is studied in [3] and [13]. In [13], the authors focus on the effect of asymmetric networks on TCP performance and show, by means of analysis and simulation, the performance degradation of TCP (Reno), due to frequent timeouts. However this study does not consider a specific MAC protocol in the reverse path (upstream channel). This model is appropriate for ADSL modems, or configurations that use a telephone line or cellular phone medium in the reverse channel where the only delay is the sum of the queuing delay and the propagation delay. However, in the case of multiple access media, such as HFC, it is important to study the effect of the MAC protocol and bandwidth reservation on TCP performance. In particular we find that TCP over HFC networks suffers from persistent gaps in the acknowledgments stream. This so-called "ACK-compression" behavior, [12], [20] is shown to degrade the TCP throughput by causing a burstiness in the TCP sending rate. We propose an algorithm that attempts to prevent ACK-compressions by preserving a constant spacing between consecutive ACKs and their corresponding data packets. We find that the proposed algorithm improves TCP performance under different offered loads and TCP data buffer sizes. The rest of the paper is structured as follows. Section 2 presents the MAC model as specified in [5]. Sections 3 describes the simulation model. Section 4 presents TCP performance results. In section 5 we propose some improvements to TCP performance over HFC. Concluding remarks are presented in Section 7. Additional details on the simulation parameters and the TCP protocol can be found in the Appendix.

2 HFC MAC Protocol Overview

The frame format of the MAC protocol defined in [5] is shown in Figure 1 . The upstream channel is divided into

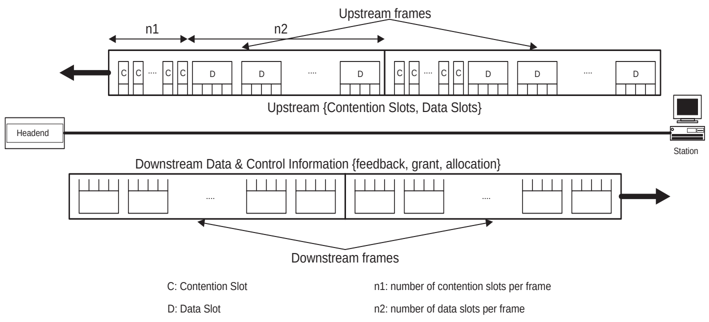

Upstream frames
n1
n2
C
D
Upstream {Contention Slots, Data Slots}
Headend
Station
Downstream Data & Control Information {feedback, grant, allocation}
Downstream frames
C: Contention Slot
D: Data Slot
n1: number of contention slots per frame
n2: number of data slots per frame

Figure 1: Frame format of 802.14 MAC protocol

minislots. Minislots that are used to transmit requests in contention are commonly referred to as Contention Slots (CS). CSs are used to request grants for data transmission (in Data Slots). A special field in the Data Slots may be used to send piggybacked requests (not subject to contention). One or more minislots can form a Data Slot (DS) to carry data packets, such as ATM cells. The size of the CS and DS is chosen so that a DS is accommodated in an integer number of minislots. In our study, we assume that data traffic generated at the station is carried via ATM cells (each DS can carry one ATM cell), conforming to the draft standard. The headend (HE) allocates the number of CS and DS slots available on the upstream channel in fixed size clusters or frames.

At start-up, the stations are ranged or synchronized with respect to the headend in order to gain a unique time reference. Each CS is assigned a Request Queue (*RQ*) value by the headend. When a station has data to transmit, it waits for a CS with an *RQ* value equal to zero, and sends a request. The station waits for feedback information about the collision status and allocation of DS contained in downstream control messages. In case of collision (two or more requests arriving at the headend at the same time), the headend performs the ternary-tree blocking algorithm [1] and assigns different CSs in the next cluster with *RQ* values greater then 0. The main principle of the ternary-tree algorithm relies in splitting all stations involved in a collision randomly into 3 subsets and then allowing stations in the various subsets to retransmit their requests sequentially. In case, the request is successfully received at the headend, the station is granted a DS to send its data. There are two grant scheduling algorithms considered in this paper: First Come First Serve (FCFS) and Round Robin (RR). For FCFS, the HE grants each station the totality of the requested slots before giving any grants to other stations. For RR, the grants are distributed to stations in a Round Robin fashion. A station sends a cell and waits for all the other stations that have successfully transmitted requests to the HE to send their data.

3 Simulation model

In this section we provide a description of the simulation environment and parameters. We use the NIST ATM Network Simulator [6] to implement TCP Tahoe and TCP Reno as described in [19]. The simulator also implements a MAC protocol for HFC networks as described in [5]. The network model considered is illustrated in Figure 2. It consists of a hybrid ATM-HFC configuration where a TCP source in the ATM network sends packets to a TCP destination in the HFC network.

A network topology is chosen with one TCP connection for a better understanding of TCP dynamics. The link capacity between the TCP source and *SW1* is set to 155 Mbits/s, while the link capacity between *SW1* and *SW2* is set to 6 Mbits/s to represent the bottleneck link. The propagation delay between *SW1* and *SW2* is set to 1.25 ms. *SW1* and *SW2* implement the Early Packet Discard strategy (EPD) [15]. The TCP source is assumed to have an infinite number of packets to send. In order to stress contention on the upstream channel, we consider 200 stations with on-off sources sending data upstream to constitute a background traffic. Sources generate fixed size 48-byte packets (encapsulated in ATM cells) according to a Poisson distribution with a mean arrival rate of $\lambda = \frac{L \times UR \times 48}{53 \times N}$. *L* is the percentage of the offered load, *UR* is the upstream channel rate and *N* is the number of stations. We use an AAL5 encapsulation of IP packets. After segmentation, cells are queued in a FIFO queue inside the MAC layer. Unlike [13], we assume that the FIFO queue inside the MAC layer is large enough to accommodate all incoming ACKs from a single TCP connection. Details on simulation parameters for both HFC and TCP can be found in Appendix A. We also provide a background on TCP protocol in Appendix B.

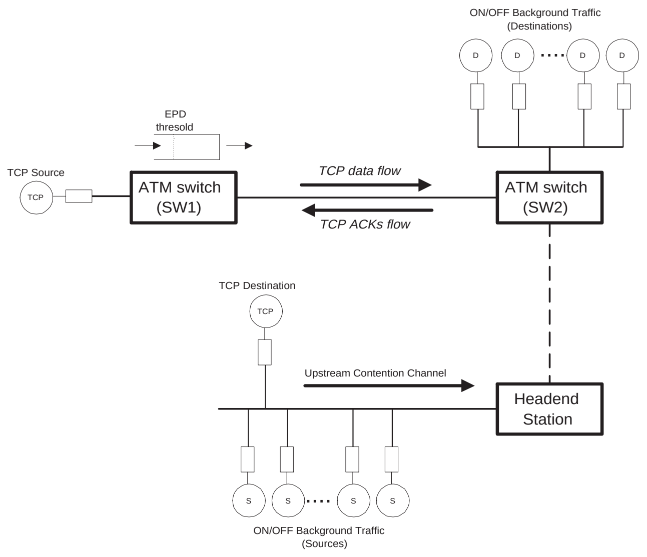


Figure 2: Simulation model

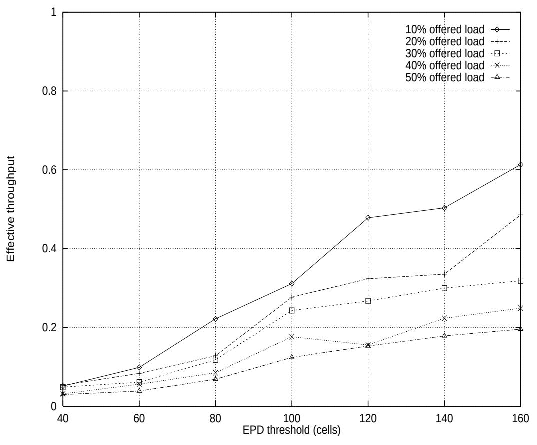
4 Performance of TCP over HFC

In this section we analyse the dynamics of TCP traffic. We consider the *effective throughput* of a single TCP connection as a function of the EPD [1] threshold in *SW1* for different upstream offered loads and grant scheduling algorithms, namely, FCFS, and RR. We define the TCP effective throughput, as the throughput that is usable by higher layer protocols [15].

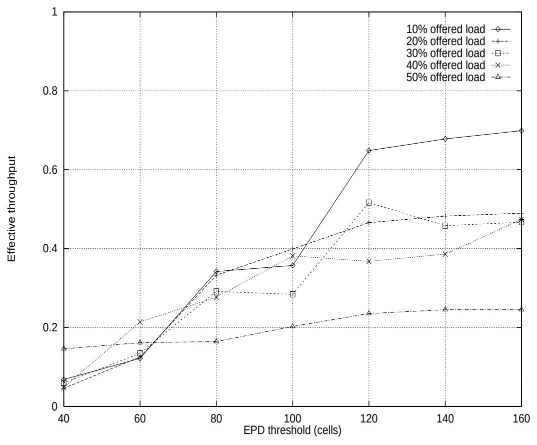
4.1 Simulation results

The effective throughput for FCFS and RR scheduling is depicted in Figure 3-(a) and Figure 3-(b) respectively.

1.we assume that the switch buffer is large enough to ensure that if the first cell of a packet is accepted (based on the EPD algorithm), all remaining cells from the same packet are queued.



(a) FCFS HE Grant Allocation



(b) RR HE Grant Allocation

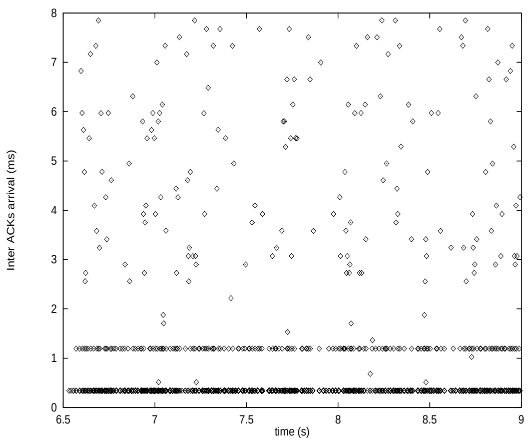
Figure 3: TCP Effective throughput vs. EPD threshold

The y-axis gives the throughput as a percentage of the maximum bottleneck link capacity (6Mbits/s). We note that it is low and can conclude that the TCP connection is unable to fill up the network pipe. The bandwidth-RTT product doesn't indicate any reason for this under-utilization. For different upstream offered loads with a maximum congestion window size of 64 Kbytes, we find that the maximum window size is greater than all mean products (with the exception of 50% load).

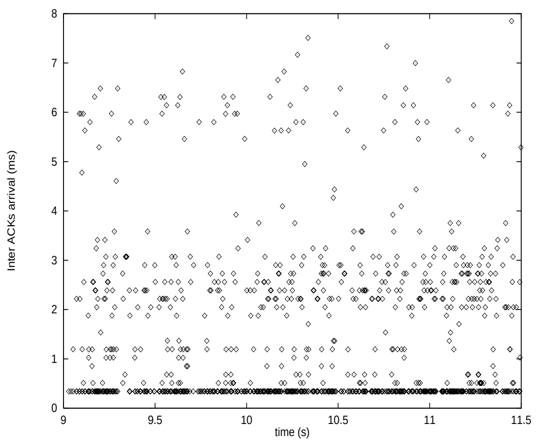
4.2 Results analysis

TCP congestion window suggests that the source suffers from frequent timeouts and fast retransmit fast recovery periods. One possible reason is the ACK compression behavior usually observed when ACKs encounter non empty queues. When ACKs leave the bottleneck buffer their spacing is smaller than the original spacing at queue entry [2] [20].

To confirm this problem we plot the interarrival time of consecutive ACKs for 30% offered load with FCFS and RR in Figure 4-(a) and Figure 4-(b). In Figure 4-(a), we can identify two groups of ACK interarrival times at y=0.34ms and y=1.19ms. The interval between two data packets sent back to back, is given by *Packet_size*/μ_1 = 1.55 ms where μ_1 is the forward link capacity. This is also equal to the minimum spacing between two TCP data packets. In our simulation we assume that the time to process a TCP packet and send its corresponding ACK is constant. Thus the spacing between ACKs should be equal to the spacing between their corresponding data packets if we look for a good performance. However, Figure 4-(a) and Figure 4-(b) the spacing between them is less than the minimum spacing between the data packets. This confirms that ACK compression is the main reason for the low effective



(a) FCFS HE Grant Allocation



(b) RR HE Grant Allocation

Figure 4: ACK Interarrival Time, upstream load=30%

throughput observed.

A look on the MAC layer helps us determine that the ACK-compression is the result of the grant allocation mechanism used to send ACKs at the MAC layer. The interval between two ACKs is equal to *ACK_size*/*Rate*$_{upstream}$, where *ACK_size*= *N*$_{DataSlot}$**Size*$_{DataSlot}$. Values of inter ACK spacing equal to 2*64*8/3000=0.34ms (Figure 4-(a) and Figure 4-(b) correspond to the case where two ACKs are sent back to back in the same frame (Figure 6-(a)). The second group of points at *y*=1.19ms corresponds to ACKs being sent back to back in two different frames. In this case, the spacing between two ACKs is equal to two DSs (corresponding to an ACK packet) and *n1* CS (Figure 1). Since each DS corresponds to four CSs, the spacing is given by 7*64*8/3000=1.19ms. Note that each ACK packet requires two DSs and there are 20 CS (or 5 DSs) in each cluster for a total of 7 DSs. This is illustrated in Figure 6-(b) and Figure 6-(c). With RR scheduling, we note that the minimum ACK spacing is also equal 0.34ms when the request queue at the HE contains only one request from the TCP receiver station. Figure 5 shows the distribution of ACK spacing for both FCFS and RR. More than 85 % of ACK spacing is lower than the minimum spacing between packets for FCFS. For RR scheduling, the percentage of ACKs below 1.55 ms is relatively low since the RR algorithm introduces spacing during scheduling. This explains why better throughput is obtained in Figure 3-(b) with RR than with FCFS in Figure 3-(a).

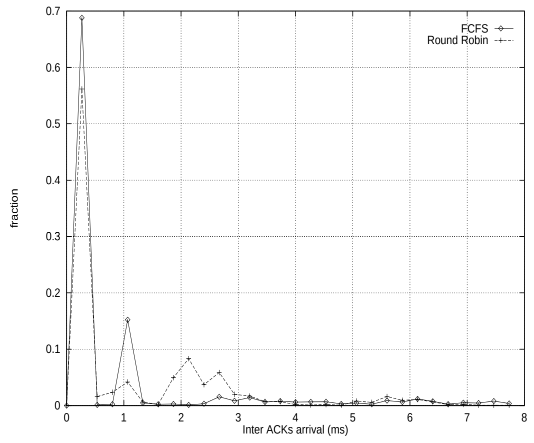


Figure 5: Distribution of ACK Interarrival time for FCFS and RR, upstream offered load=30%

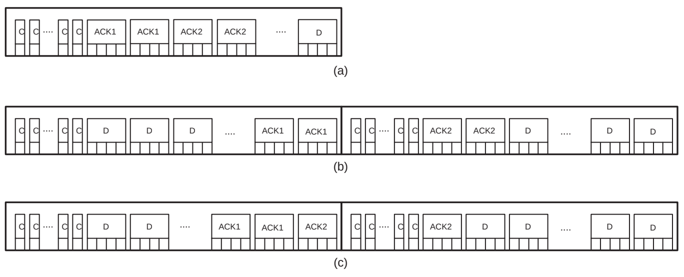


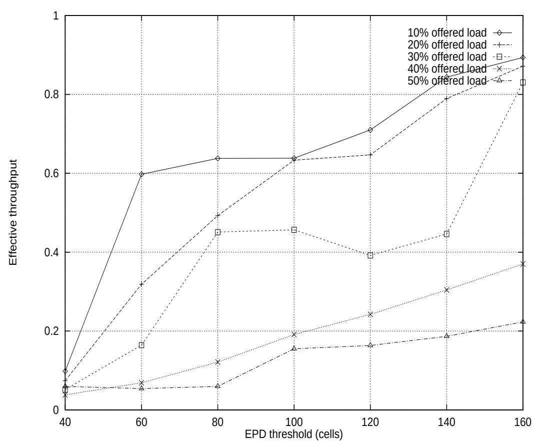
Figure 6: ACK compression scenarios

5 Solutions to the ACK compression behavior

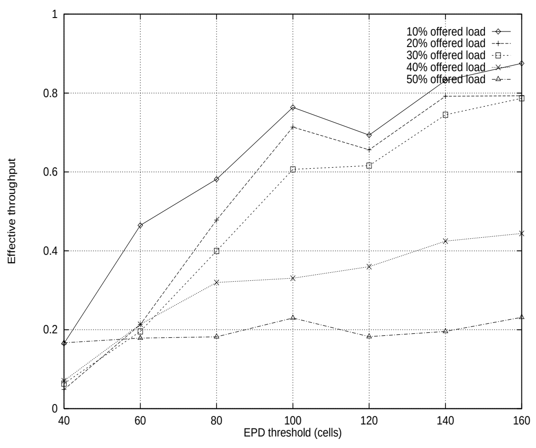
We discuss possible improvements to this problem. As explained in Section 4, TCP low throughput is mainly attributed to ACK compression. Solutions vary in complexity and implementation. In [12], a general solution is proposed with separate queues for ACKs and data respectively (to improve TCP performance over ATM in the case of two-way traffic). This ensures that ACKs are not subject to delay and delay variation caused by TCP data packets. Another improvement is to prevent ACKs from being subject to delay and delay variation inside the MAC layer. For example, the HE can predict the number of data slots needed by a station to send its ACK packets based on the number of TCP packets it forwards in the downstream direction. Such a solution requires processing of each ATM cell header in order to ensure that the ATM cell is part of a data packet. It requires a good estimator for the RTT (between the HE and the station) and the processing delay at the HE. Although this method should be further investigated we focus on other mechanisms for ACK spacing implemented in the station MAC layer with less complexity. We first investigate the performance of TCP over HFC using piggybacked requests. Then we propose an algorithm for ACK spacing that significantly increases TCP throughput.

5.1 TCP performance using Piggybacking

Piggybacking in the MAC layer consists in sending requests for additional data transmission along with a data packet without having to go through contention. After the first request (sent in contention), subsequent requests are transmitted in the Extended Bandwidth Request (EBR) field of the MAC data PDU. Since the request is piggybacked in the DS, it is not subject to contention. We plot in Figure 7-(a) and (b) the effective throughput of a TCP connection, using piggybacking as a function of the EPD threshold for FCFS and RR respectively. Note that in order to keep the offered load comparable to the simulations performed in Section 4, we only use piggybacking for the transmission of TCP ACK packets (i.e. not for background traffic). As a direct result of piggybacking the effective



(a) FCFS HE Grant Allocation



(b) RR HE Grant Allocation

Figure 7: TCP effective throughput vs. EPD threshold, piggybacked requests

cells).We note again that RR scheduling gives better performance results than FCFS scheduling. We attribute this to the spacing introduced by the RR scheduling that increases the probability to send requests for additional ACK transmissions in every DS granted.

From the results obtained, we can conclude that piggybacking reduces the contention percentage in the case of multiple TCP sources and gives a shorter upstream delay. However, the throughput for 50% offered load remains low due to the relatively large delay incurred. In Figure 8, we plot the interarrival distribution of TCP ACKs. Compared to Figure 5, we note a reduction in the ratio of ACKs sent back to back: with a spacing equal to 0.34ms. Moreover, a relatively large number of ACKs arrives with a spacing greater than 1.55ms (which is the minimum spacing between data packets) and hence further alleviates the ACK compression phenomenon reported.

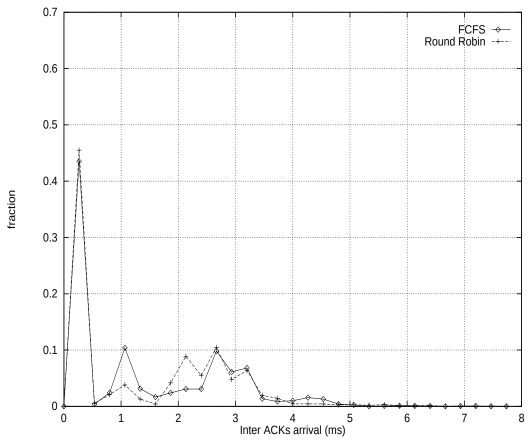


Figure 8: Distribution of ACK Interarrival Time for FCFS and RR, Upstream offered load = 30%

5.2 Acknowledgments spacing

Piggybacking does not adapt itself to a situation where the offered load periodicity (rate) is different from the arriving packet rate (it does not conserve the optimal spacing between acknowledgments). In this case piggybacked requests arrive at the headend with a rate (relative to the offered load) that does not match the packet rate and thus lead to bandwidth waste. We present now a new algorithm that dynamically controls the spacing between TCP

ACK. This interval is chosen as to increase the bandwidth and to prevent packet losses. We first describe the algorithm and give a pseudo code description. We then explain the simulation results and compare the two solutions.

5.2.1 Dynamic rate tracking algorithm

The algorithm tracks the bottleneck rate capacity μ_1. It then calculates the adequate ACK rate μ_p for the measured μ_1.

$\mu_p = \mu_1 / Packet\_size$

Different methods may be used to estimate the bottleneck rate. Some have been already proposed in rate based flow control algorithms [11], [16]. We use a very simple method well adapted to TCP behavior in SlowStart and Congestion-Avoidance working regions. It measures the minimal interarrival time, τ, between two consecutive ACKs to decide of the ACK sending rate (spacing). In this case $1/\tau$ is an approximation of the bottleneck rate, i.e. μ_1. Since each ACK corresponds to two ATM cells (for Classical IP over ATM), the requested spacing is then set to $\tau/2$. As we said this is adapted to TCP, since it usually sends packet pairs in both Slow-Start and Congestion-Avoidance regions. We can measure and always use the shortest interarrival time between two consecutive ACKs (in the MAC layer FIFO queue). The pseudo code for the proposed algorithm is as follows:

```
When a new ACK is received
    if MAC_queue_is_not_empty()
        if $\tau$==0
            $\tau$ = clock() - last_ack_arrival
        else
            $\tau$ = filter (clock()    - last_ac  k_arrival,  $\tau$)
            /* filter can be Min, Max or Mean */

    last_ack_arrival = clock()

Each time a request for new data transmission is sent:
    request queue_size() grants with a t      ransmission rate    $\tau$/2
    $\tau$=0
```

The proposed algorithm can be implemented using an additional option in the HFC MAC layer that permits to specify an interval for transmitting data. This is achieved at the cost of a slight increase in the MAC grant request PDU size, namely an 8-bit field containing the transmission rate.

5.2.2 Simulation Results

Figure 9 gives the effective throughput of TCP over HFC using the ACK spacing algorithm proposed. We note that it improves TCP performance for different offered loads. As expected, no improvement can be achieved beyond a load of 50%. This is due to the large bandwidth delay product incurred at this load.

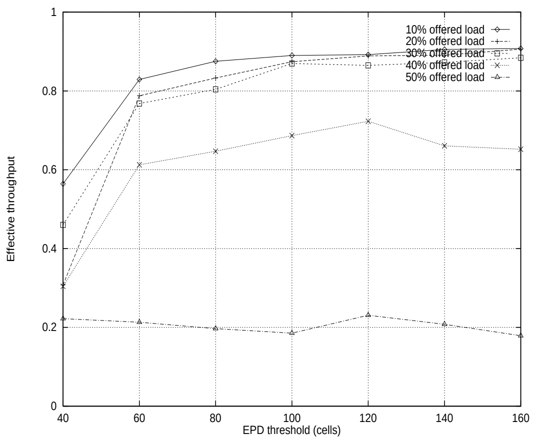


Figure 9: Effective throughput vs EPD threshold, min spacing

5.2.3 Piggybacking vs. ACK spacing

As we have mentioned at the beginning of this paragraph, although piggybacking reduces the ACK compression behavior, by introducing some kind of spacing on successive ACKs, its spacing can not match the original spacing between ACKs. In this situation ACKs spacing can result in ACK compression and bad performance. To show this shortcoming, we simulated different bottleneck rates (link capacity between SW1 and SW2): 1.5 Mbits/s from 0 to 10 s, 2 Mbits/s from 10 to 20 s and 2.5 Mbits/s from 20 to 30 s of the simulation time (see Figure 10). Our goal is to compare the stability of piggybacking and ACK spacing. We found that ACK spacing algorithm prevents from TCP timeouts for the three bottleneck rates. For piggybacking, while the performance is similar to ACK spacing when the bottleneck rate is under 2.5 Mbits/s, many timeouts are observed when the bottleneck rate is set to 2 Mbits/s and 1.5 Mbits/s. For 1.5 Mbits/s the throughput degradation is very visible (large periods of inactivity). The spacing introduced by piggybacking does not always match the bottleneck rate, and ACKs can be subject to compression.

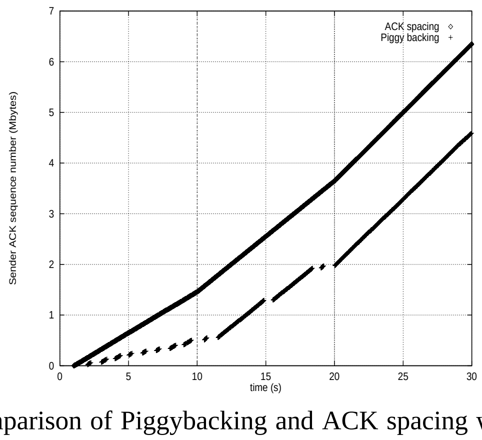


Figure 10: Comparison of Piggybacking and ACK spacing with different bottleneck rates: 1.5, 2, 2.5 Mbits for 20% offered load, EPD threshold = 100 cells

6 Concluding Remarks

This paper gives some performance results and improvements of the TCP protocol over HFC networks. First, we show that poor TCP performance is observed due ACK compression. We compare the performance of FCFS and RR HE scheduling algorithms and found that RR scheduling results in better performance due to the "natural" spacing introduced on successive TCP ACKs. Second, we investigate the effect of piggybacking on TCP performance and found that it reduces the delay and the delay variation of TCP ACKs. As a result TCP efficiency is improved in all cases. However piggybacking can still result in bad performance when the data path capacity is small. Finally, an algorithm for ACK spacing is proposed and is shown to give optimal performance for different offered loads and network buffer sizes. This algorithm is very simple and aims at conserving the ACKs interarrival time, to respect the TCP self-clocking mechanism. It can be generalized for other applications, since even if packets are subject to delay (due to grant requests), it is desirable to conserve, at the MAC layer, the spacing introduced by the applications. We believe that more studies need to be led in order to investigate other issues such as TCP fairness and delay over HFC. Although, we used TCP-RENO in our simulations to reflect the large majority of TCP/IP stacks, it may be interesting to examine the performance of different TCP algorithms, such as SACK-TCP and New-Reno.

7 References

[1] C. Bisdikian. “msStart: A Random Access Algorithm for the IEEE 802.14 HFC Network,” *Technical Report, RC 20466*, IBM Research Division, T.J. Watson Research Center, June 1996.

[2] J.C. Bolot, “Charaterizing End-to-End Packet Delay and Loss in the Internet” , *Journal of High-Speed Networks*, vol. 2, no. 3, pp. 305-323, December 1993.

[3] O. Elloumi, H. Afifi, M. Hamdi. “Improving Congestion Avoidance Algorithms in Asymmetric Networks”. Proc. IEEE ICC ’97. Montreal. June 1997.

[4] J. C. Hoe, “Improving the Startup Behavior of a Congestion Control Scheeme for TCP”, Proc. ACM SIGCOMM’96, August 1996, pp 270-280, Stanford, CA.

[5] IEEE 802.14 Working Group, Media Access Control, IEEE Draft Std. 802.14, Draft 2 R2, July 1997.

[6] N. Golmie, A. Koenig, and D. Su, “The NIST ATM Network Simulator, Operation and Programming,” Version 1.0, *NISTIR 5703*, March 1995.

[7] N. Golmie, S. Masson, G. Pieris, and D. Su:“Performance Evaluation of MAC Protocol Components for HFC Networks,” Proceedings of the International Society for Optical Engineering, Photonics East’96 Symposium, 18-22 November 1996, Boston, Massachusetts. Also appeared in Computer Communication, June 1997.

[8] N. Golmie, S. Masson, G. Pieris and D. Su:“Performance Evaluation of Contention Resolution Algorithms: Ternary-tree vs p-Persistence,” *IEEE 802.14 Standard Group*, IEEE 802.14/96-241, October 1996.

[9] N. Golmie, M. Corner, J. Liebherr, D. Su, “Improving the Effectiveness of ATM Traffic Control over Hybrid Fiber-Coax Networks”, Proc. of IEEE Globecom 1997, Phoenix, Arizona.

[10] V. Jacobson, “Congestion Avoidance and Control”, Proc. ACM SIGCOMM’88, pages 314-329, August 1988.

[11] R. Jain, “Rate Based Flow Control”, Proc. M SOMM’96, Aust b.c. 22, pp 270-280, Leonides, Gr.

[12] L. Kalampoukas, A. Varma, K. K. Ramakrishnan, “Two-Way TCP Traffic over ATM: Effects and Analysis”, Proc. Infocom’97, April 1997, Kobe, Japan.

[13] T. V. Lakshman, U. Madhow, B. Suter, “Window-based error recovery and flow control with a slow acknowledgment channel: a study of TCP/IP performance”, Proc. Infocom’97, April 1997, Kobe, Japan.
[14] J. Postel, “Transmission control protocol”, Request for comment 793, DDN Network Information Center, SRI International, September 1981.
[15] A. Romanow, S. Floyd, “ Dynamics of TCP Traffic over ATM Networks”, IEEE JSAC, V. 13 N. 4, May 1995, p. 633-641.
[16] S. Keshav, “Packet Pair Flow Control”, to appear in IEEE/ACM Transactions on Networking, available from http://www.cs.cornell.edu/skeshav/papers.html
[17] V. Paxson, “Measurements and Analysis of End-to-End Internet Dynamics”, Ph.D. Thesis, LBNL-40319, UCB//CSD-97-945, University of California, Berkley.
[18] W. R. Stevens, “TCP/IP Illustrated, volume 1”, Addison-Wesley Publishing Compagny, 1994.
[19] W. R. Stevens, “TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, Request for Comments 2001, January 1997.
[20] L. Zhang, S. Shenker, D. D. Clark, “Observations on the dynamics of a congestion control algorithm: the effects of 2-way traffic”, Proc. ACM SIGCOMM’91, pp 133-147, Sept 1991, Zurich, Switzerland.

Appendix A: Simulation parameters

Table 1 : MAC Model parameters

Simulation parameter	Value
Number of active stations	200
Distance from nearest/furthest station to headend	25/200 km
Upstream data transmission rate	3 Mbits/s
Downstream data transmission rate	30 Mbits/s
Propagation Delay	5 µs/km
Length of Simulation Run	15 s
Length of run prior to gathering statistics	1 s
Guardband and preamble between transmissions	Duration of 5 bytes
Data Slot Size	64 bytes
Contention Slot Size	16 bytes
DS/CS Size Ratio	4:1
Cluster Size	2.27 ms
Maximum Request Size	32
Headend Processing Delay	1 ms

Table 2 : TCP parameters

Simulation parameter	Value
Maximum Transfer Unit (MTU)	1 KB
Timeout granularity	500 ms
Packet processing time	100 µs
Congestion avoidance algorithm	TCP Reno
Maximum congestion window size	64 KB
Initial SSThreshold	8 KB

Appendix B: TCP Protocol Background Information

Most of today's internet applications use the TCP protocol as defined in [14]. In this section, we describe two basic concepts of TCP related to congestion avoidance and control.

TCP Congestion Avoidance and Control Mechanisms have significantly evolved in the past few years although the protocol packet format and its state machine have remained unchanged. Most versions of TCP control mechanisms aim at improving the estimation of available network bandwidth and preventing timeouts in order to maintain stability and throughput.

The slow-start algorithm [10] was proposed by Jacobson as a congestion avoidance and control algorithm for TCP after a congestion collapse of the Internet. This algorithm introduces a congestion window mechanism to control the number of bytes that the sender is able to transmit before waiting for an acknowledgment. For each received acknowledgment, two new segments are sent. When the window size reaches a threshold value, SSThreshold, the algorithm operates in Congestion Avoidance mode. The slow start is triggered every retransmission timeout by setting SSThreshold to half the congestion window and the congestion window to one segment. In the Congestion Avoidance phase, the congestion window is increased by one segment every Round Trip Time (RTT). Thus when the mechanism anticipates a congestion, it increases the congestion window linearly rather than exponentially. The upper limit for this region is the value of the receiver's advertised window. If the transmitter receives three duplicate acknowledgments, SSThreshold is set to half of the preceding congestion window size while this latter is set to one packet for TCP Tahoe and half of the previous congestion window for TCP Reno. At this point the algorithm assumes that the packet is lost, and retransmits it before the timer expires. This algorithm is known as the fast-retransmit mechanism.

TCP Tahoe experiences low throughput when packets are lost because of the slow-start algorithm. TCP Reno performs better in the case of single packet loss within one window of data because the congestion window is decreased by half rather than set to one. However, in the case of multiple packet losses, TCP Reno also experiences low throughput since it can easily be subject to timeouts leading to long idle periods. This is outlined in [4] and [13]. In [4], the author proposes the so-called ``New-Reno" algorithm to avoid this problem.

TCP Self-clocking Principle [10] estimates the bottleneck bandwidth by letting the sender rate exactly match the available bandwidth along the network path. Thus if the time to process packets at the receiver is constant, ACKs are spaced according to the bottleneck in the forward path. If the network is symmetric, the ACK spacing is preserved in the backward direction since the ACK packet size is much smaller than the data packet size and thus less likely to encounter congestion. When the sender transmits a packet for each arriving ACK, the packet sending rate matches the bottleneck service rate and this constitutes a ``self-clocking" mechanism, an ``idealized state" as mentioned in [12], [17] and [20].

However, this mechanism fails in the case of delay variations in both the forward and backward directions. In particular, in the case of two-way TCP traffic, ACK spacing is not preserved due to the interaction between data packets and ACKs: ACKs accumulate behind large data packets and then leave the bottleneck with a smaller spacing than the spacing corresponding to their data packets [12], [20]. Furthermore, ACK compression may lead to a rapid network queue build-up and a high packet loss percentage as shown in [20]. Even with infinite buffers, the network utilization is expected to drop considerably.